Coloquio Programación en FORTRAN

Día 5 (clases 9 y 10).

- Paralelismo con OpenMP.
- Una mirada a los métodos de computación paralela.

OpenMP

- OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida.
- Está estandarizado y se implementa en los compiladores. Se activa con flags: -openmp (Intel), -fopenmp (GNU).

  Se escriben directivas en el codigo fuente.

  No confunda ni compare OpenMP con OpenMPI.

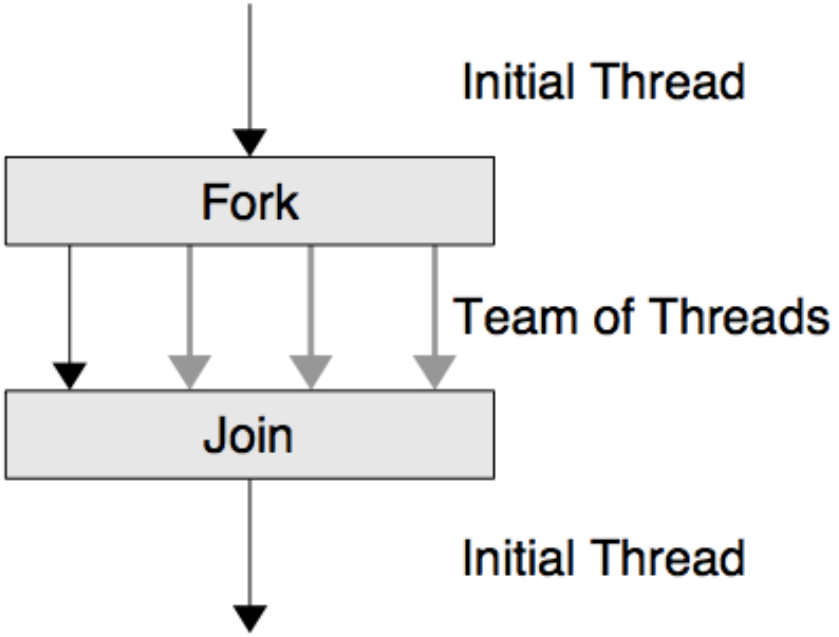
OpenMP

- Ver el programa Paralelismo/test.f para tener idea de como luce.

```
        program test
        INTEGER OMP_GET_NUM_THREADS,OMP_GET_THREAD_NUM,TID
        INTEGER*8 I,S
        S=0
      print *,
'**********************************************************'
      print *, 'PROGRAMA PARALELO'
      print *,
'**********************************************************'
!$OMP PARALLEL DEFAULT(NONE) PRIVATE(I,TID,FS)
SHARED(S,NTHREADS)
      TID = OMP_GET_THREAD_NUM()
      IF (TID.EQ.0) THEN
         NTHREADS=OMP_GET_NUM_THREADS()
         WRITE (*,*) 'Numero de hilos  ',NTHREADS
   ...
```

OpenMP

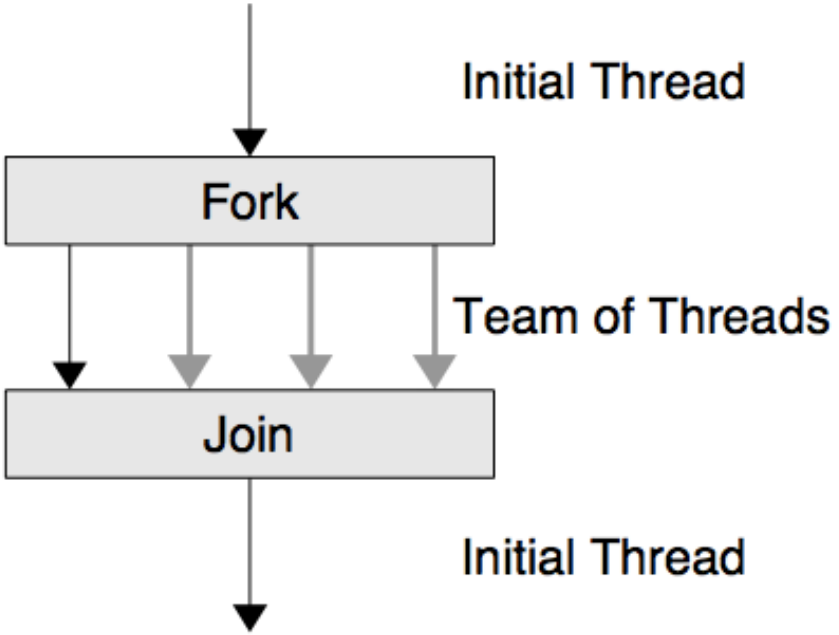
- Hilo (thread) : proceso que ejecuta una secuencia de programa. Los programas en serie tienen un solo hilo.
- OpenMP permite crear hilos que trabajan cooperativamente y aniquilarlos cuando no se necesitan.



OpenMP: creando hilos

Definicion de un segmento paralelo del programa: creación y terminacion de hilos.

```
!$OMP PARALLEL
    Seccion paralela
!$OMP END PARALLEL
```



¿Cuantos hilos se crean ?

```
export OMP_NUM_THREADS=4
```

► Hace que se creen 4 hilos

Si no se define OMP_NUM_THREADS, el sistema operativo lo define según el numero de procesadores lógicos. Ver /proc/cpuinfo

OpenMP: creando hilos

Se puede definir explicitamente o limitar la cantidad de hilos, mediante cláusulas de la directiva OMP PARALLEL.

Formato general en FORTRAN

```
!$omp parallel [clause[[,] clause]...]
   Seccion de programa
!$omp end parallel
```

En C/C++

```
#pragma omp parallel [clause[[,] clause]...]
  {
Seccion de programa
  }
```

Cláusulas de PARALLEL

if *(scalar-expression)*	(C/C++)
if *(scalar-logical-expression)*	(Fortran)
num_threads *(integer-expression)*	(C/C++)
num_threads *(scalar-integer-expression)*	(Fortran)
private *(list)*	
firstprivate *(list)*	
shared *(list)*	
default(none|shared)	(C/C++)
default(none|shared|private)	(Fortran)
copyin *(list)*	
reduction *(operator:list)*	(C/C++)
reduction *({operator|intrinsic_procedure_name}:list)*	(Fortran)

OpenMP: paralelizando bucles

```
!$OMP PARALLEL
!$OMP DO
do 10 j = 1, jmax i=1
      vv (i,j) = v (i,j,m)
      do 60 i = 2, imax-1
         vv (i,j) = vv (i-1,j) + b(i,j)
 60   continue
      i = imax
      vv (i,j) = vv(i-1,j)
      do 100 i = imax-1, 1, -1
         vv (i,j) = vv (i+1,j) + a (i,j)
 100  continue
 10  continue
!$OMP END DO
!$OMP END PARALLEL
```

Creación de hilos

Distribución del bucle exterior entre los hilos

Los bucles internos no se distribuyen

Sincronización

OpenMP: paralelizando bucles variables privadas

Por defecto todos los hilos acceden a toda la memoria y a todas las variables del programa.

La variable j del ciclo anterior es privada. Cada hilo tiene su valor.

```
!$omp do
do j=1, 100
   a(j)= …...
   ...
end do
!$omp end do
```

OpenMP: paralelizando bucles variables privadas

Por defecto todos los hilos acceden a toda la memoria y a todas las variables del programa.

La variable j del ciclo anterior es privada. En cada hilo j tiene su propio valor.

```
!$omp do shared(a) private(j)
do j=1, 100
   a(j)= …...
   ...
end do
!$omp end do
```

OpenMP: paralelizando secciones sin bucles

Construcción SECTIONS

```
!$omp sections [clausulas]
    !$omp section
      bloque de programa
    !$omp section
      otro bloque
      …
!$omp end sections [nowait]
```

OpenMP: serializando

Construcción SINGLE : Especifica una sección para ser ejecutada por un solo hilo. Por ejemplo, una operación de entrada/salida

```
!$omp single [clausulas]
     ...
!$omp end single [nowait, [copy private]]
```

OpenMP: arreglos

Construcción WORKSHARE : Se aplica a operaciones con arreglos de FORTRAN.

Ejemplo

```
!$omp parallel shared(a,b,c,n)
  !$omp workshare
  b(1:n)=b(1:n) +2
  c(1:n)=b(1:n) +1
  a(1:n)=b(1:n)+c(1:n)
  !$omp end workshare
!$omp end parallel
```

OpenMP: compilacion

Ejemplo:

gfortran -fopenmp -O3 -o binario programa.f90

Después de ejecuta

./binario

Sin la opcion de optimizacion -O3 puede que el programa paralelo sea mucho mas lento.

Unicidad de la fuente

Todas las directivas !$OMP … son interpretadas como comentarios por el compilador, si no se le llama con el flag específico (-fopenmp para GNU).

Sin embargo, hay funciones específicas que se usan para controlar el programa y no son directivas.

```
      integer :: TID
      TID=0
!$    TID=omp_get_thread_num()
      If (TID==0) then
       ….
      else
      ….
```

Si no se pone !$ al compilar sin -fopenmp daria una error de una función no resuelta.

Unicidad de la fuente

Otro mecanismo para mantener un codigo fuente único es el de las opciones de preprocesamiento.

```
#ifdef __OPENMP
  INTEGER :: nth, ith, omp_get_thread_num,
omp_get_num_threads
#endif

...
!$omp parallel private( nth, ith, ew, it1, it2, z, zp,
tt, kk1, kk2, cc1, cc2, &
!$omp                   ng_2d, k1, k2, gp2, ipol, t, gp,
ff, arg1, arg2, t1, t2 )
#endif
#ifdef __OPENMP
  nth=omp_get_num_threads()
  ith=omp_get_thread_num()
#endif
```

Unicidad de la fuente

Otro mecanismo para mantener un codigo fuente único es el de las opciones de preprocesamiento.

```
#ifdef __OPENMP
  INTEGER :: nth, ith, omp_get_thread_num,
omp_get_num_threads
#endif
...
```

__OPENMP es un flag de preprocesamiento. Puede tener cualquier nombre. La mayoria de los compiladores admiten este mecanismo. El codigo entre #ifdef y #endif es compilado solamente si se invoca la opcion -Dflag, por ejemplo

gfortran -c -D__OPENMP archivo.f90

Mostrar un Makefile de un programa como Quantum Espresso.

¿Como medimos la eficacia de la paralelización?

T_1 = tiempo de ejecución serie

T_P = tiempo de ejecución con P procesos

Speedup: $S = \frac{T_1}{T_P}$

Ley de Amdahl

$$S = \frac{1}{\frac{f_{par}}{P} + (1 - f_{par})} < P$$

donde f_{par} es la fracción paralela del programa

Using OpenMP

PORTABLE SHARED MEMORY PARALLEL PROGRAMMING

BARBARA CHAPMAN,
GABRIELE JOST,
AND RUUD VAN DER PAS

foreword by
DAVID J. KUCK

Tarea 2

Resolver la ecuación de Laplace en un rectángulo de lados a y b con la condición de frontera de que el potencial tome un valor constante y diferente en cada lado.

$$V(x,0)=0, V(x,b)=V, V(0,y)=yV/d, V(a,y)=y^2V/d^2$$

El programa debe crear una malla de N_1xN_2 puntos, asigne valores aleatorios al potencial en vada punto, y recorra la malla aplicando la regla

$$V(i,j)=\frac{V(i,j+1)+V(i+1,j)+V(i-1,j)+V(i,j-1)}{4}$$

Este procedimiento debe aplicarse iterativamente hasta que los V(i,j) ya no varien mas. Una forma de hacerlo es

Tarea 2

$$\delta = 0$$

ciclo en i , j recorriendo los $N_1 \times N_2$ puntos de la malla

$$c = V(i, j)$$

$$V(i, j) = \frac{V(i, j+1) + V(i+1, j) + V(i-1, j) + V(i, j-1)}{4}$$

$$\delta V = \delta V + \frac{\left(V(i, j) - c\right)^2}{N_1 N_2}$$

cierre de ciclos en i , j

si $\frac{\delta V}{V} > 10^{-3}$, repetir el ciclo, terminar. Si no, repetir.

Haga un programa eficiente, sin utilizar una biblioteca de solución de de ecuaciones diferenciales. Haga el programa eficiente, y paralelícelo con OpenMP. Determine el speedup en función del numero de hilos.

Otros modelos de paralelismo

- Pipeline. Usar varias partes del hardware para hacer varias operaciones en el mismo ciclo de reloj. Los compiladores hacen varias optimizaciones.

$$c_1 = a_1 + b_1$$
$$c_2 = a_2 + b_2$$
$$c_3 = a_3 + b_3$$
$$c_4 = a_4 + b_4$$

```
do i=1, 4
 c(i)=a(i)+b(i)
end do
```

Años 70-90: computadores vectoriales: Ejemplo: Cray-1. El concepto está integrado en los procesadores modernos: MMX, SSE, AVX.

```
less /proc/cpuinfo
```

Modelos de paralelismo

- Varios CPU con acceso a memoria compartido entre todos por igual.

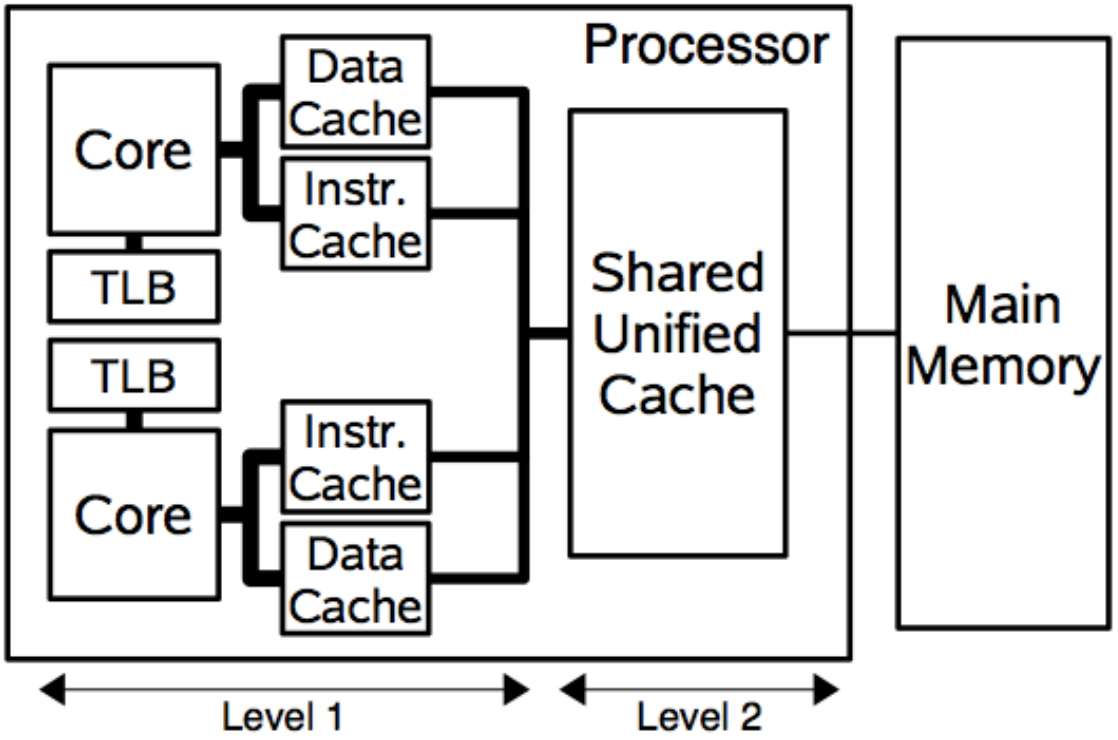
  SMP=Symmetric Multi-Processing.

  Desde los años 80 en supercomputadores.

  Commodity en el siglo XXI: Dual Pentium, Xeon, Opteron, POWER.

Modelos de paralelismo

- Multicore. Varias unidades de proceso en el mismo chip, compartiendo la memoria principal.



Cada unidad de proceso tiene acceso exclusivo a pequeñas partes de la memoria (cache). Se sigue usando el nombre SMP.

Modelos de paralelismo

- Memoria distribuida. Se unen distintos computadores mediante una red. (Massively Parallel Processing=MPP). Los componentes pueden ser de diseño específico ó *off-the-shelf*.

  Si los componentes son *off-the-shelf*, el computador se llama cluster beowulf.

  Cada unidad de proceso tiene acceso a su propia memoria, no acceso directo a la memoria de las demás unidades.

Modelos de paralelismo

- Memoria distribuida + hardware costoso que permite acceso a toda la memoria

  DSM=Distributed Shared Memory.

  Ej.: SGI Origin, SGI Altix.

Formas de hacer programas paralelos

- Instrucciones específicas de manejo de memoria. Programas específicos para cada arquitectura. Ya es obsoleto, derivó en la creación de bibliotecas estándares.
- Flags de compilación. Utilidad limitada, pero vale la pena probarlo. -ftree-parallelize-loops=n, -ftree-loop-vectorize, -floop-parallelize-all

  La utilidad es limitada porque el compilador no tiene información de los valores que tomaran las variables.Es más efectivo que el programador lo diseñe.

Formas de hacer programas paralelos

- MPI=Message Passing Interface. Es un grupo de estándares (actualmente MPI-2) diseñado para hardware de memoria distribuida.

  Conjunto de subprogramas que hacen la coordinación y la transferencia de datos entre procesos.

  Se implementa en bibliotecas: MPICH, LAM, OpenMPI, IntelMPI, etc.

  Alta dificultad de programación. Requiere una organización del programa orientada al paralelismo.

Formas de hacer programas paralelos

- OpenMP. Es también un estandar y se implementa en los compiladores. Se activa con flags: -openmp (Intel), -fopenmp (GNU).

  Se escriben directivas en el codigo fuente. Es menos eficiente que MPI, pero más fácil de programar. Un programa en serie se puede paralelizar progresivamente.

  No confunda ni compare OpenMP con OpenMPI.

  OpenMP vs MPI (LAM,MPICH,OpenMPI,...)

¿Como identifico un programa con MPI?

- Se compilan con mpif77, mpif90, mpicc.

  Los anteriores son wrappers alos compiladores g77, gfortran, y gcc, que enlazan con las bibliotecas de MPI.

- Se ejecutan con la orden

  mpirun -np *nprocs* programa

- Código fuente incluye llamadas a funciones como `CALL MPI_SEND, CALL MPI_RECV, MPI_Init, MPI_Bcast.`

Otras formas de hacer programas paralelos

- POSIX threads (Pthreads). Es un concepto parecido a OpenMP, desarrollado por la IEEE. Implementado como una biblioteca para lenguage C.